Ingeniería Informática

PROYECTO FIN DE CARRERA

Diseño y desarrollo de una aplicación de gestión de simuladores de conducción para dispositivos móviles

Autor: José Luis Nieto Torres

Tutor: Francisco Javier García Blas

Leganés, septiembre de 2017

AGRADECIMIENTOS

A mis padres, por estar siempre ahí, por su paciencia desde que terminé todas las asignaturas hasta que ha llegado este momento.

A mis tíos y mi abuela, por preguntarme siempre que nos vemos que cómo va el proyecto.

A mis amigos y compañeros de trabajo, porque me tienen que ver todos los días y por aguantarme los días buenos y los días malos.

A mi tutor Francisco Javier, por ayudarme en todo lo que ha podido y más, sobre todo en los momentos finales.

Y por último pero no por ello menos importante, a la comunidad F1Champs, porque al final esto es para ellos.

RESUMEN

Los simuladores de conducción han evolucionado de tal manera que ya no son considerados juegos. Con un nivel muy alto de realismo en cuanto a la simulación de físicas, han llegado hasta a ser usados por equipos de carreras reales. Además, en los últimos años se han visto más popularizados como eSports y atraen cada vez a más personas.

Por otro lado, las aplicaciones móviles son usadas por millones de personas en todo el mundo todos los días gracias los teléfonos inteligentes, tablets y demás dispositivos.

En este documento se van a combinar ambos mundos mediante el análisis, diseño e implementación de un Hud (heads up display) en forma de aplicación móvil Android para el simulador rFactor 2.

Palabras clave: simulador de conducción, rFactor 2, Android, Hud.

ABSTRACT

Racing simulators have evolved in such a way that they are no longer considered videogames. With a very high level of realism in physics simulation, they have even been used by real racing teams. In addition, in recent years they have been more popularized as eSports and attract more and more people.

On the other hand, mobile applications are used by millions of people around the world every day thanks to smartphones, tablets and other devices.

This document will combine both worlds by analyzing, designing and implementing a Hud (heads up display) in the form of an Android mobile application for the rFactor 2 simulator.

Keywords: racing simulator, rFactor 2, Android, Hud.

ÍNDICE GENERAL

1. INTRODUCCIÓN 11
   1.1. MOTIVACIÓN 11
   1.2. OBJETIVOS 12
   1.3. ESTRUCTURA DE LA MEMORIA 12
2. ESTADO DEL ARTE 14
   2.1. ANDROID 14
      2.1.1. Anatomía de una aplicación Android 17
      2.1.2. Ciclo de vida de una aplicación Android 19
   2.2. RFACTOR 2 21
      2.2.1. Sistema de plugins 22
3. ANÁLISIS DEL SISTEMA 25
   3.1. DIAGRAMA DE ESTADOS DEL SIMULADOR 25
   3.2 CASOS DE USO 26
      3.2.1 Descripción gráfica de los casos de uso 27
      3.2.2. Descripción textual de los casos de uso 27
   3.3. REQUISITOS DE USUARIO 29
      3.3.1. Requisitos de capacidad 30
      3.3.2. Requisitos de restricción 32
   3.4. REQUISITOS DE SOFTWARE 32
      3.4.1. Requisitos funcionales 32
      3.4.2. Requisitos no funcionales 36
   3.5. REQUISITOS DE HARDWARE 37
4. DISEÑO DEL SISTEMA 38
   4.1. ARQUITECTURA DEL SISTEMA 38
      4.1.1. Arquitectura Modelo-Vista-Controlador 38
      4.1.2. MVC en una aplicación Android 39
   4.2. DISEÑO DETALLADO 40
      4.2.1. Sistema de comunicación 42
   4.3. OTRAS DECISIONES DE DISEÑO 43

5. IMPLEMENTACIÓN Y PRUEBAS 45
5.1. IMPLEMENTACIÓN 45
5.2. PRUEBAS 48
5.2.1. Hardware de prueba 48
5.2.2. Pruebas de aceptación 49
5.3.2. Matriz de trazabilidad RSF-PA 51
6. CONCLUSIÓN Y AMPLIACIONES FUTURAS 52
6.1 CONCLUSIÓN 52
6.2. OBJETIVOS CUMPLIDOS 53
6.3. PRESUPUESTO 53
6.3.1. Costes de personal 53
6.3.2. Costes de hardware 54
6.3.3. Costes de software 54
6.3.4. Coste total 54
6.3.5. Beneficios 55
6.4. AMPLIACIONES FUTURAS 55
7. BIBLIOGRAFÍA 56

ÍNDICE DE FIGURAS

Figura 1. Arquitectura de Android 15
Figura 2. Distribución de versiones de Android en mayo de 2017 16
Figura 3. Ciclo de vida de una Actividad Android..... 20
Figura 4. Diagrama de estados del simulador..... 25
Figura 5. Arquitectura MVC del sistema..... 39
Figura 6. Diagrama de secuencia de la creación de una interfaz gráfica..... 47

ÍNDICE DE TABLAS

Tabla 1. Versiones de Android 15
Tabla 2. Fichero InternalsPlugin.hpp 22
Tabla 3. Clase InternalsPlugin 23
Tabla 4. Fichero Example.hpp 24
Tabla 5. Fichero Example.cpp 24
Tabla 6. Plantilla de Casos de Uso 26
Tabla 7. CU-01, Conectar con el simulador 28
Tabla 8. CU-02, Cambiar la interfaz del Hud 28
Tabla 9. CU-03, Actualizar telemetría 28
Tabla 10. CU-04, Actualizar sesión 29
Tabla 11. Plantilla de los requisitos de Usuario 29
Tabla 12. RUC-01 30
Tabla 13. RUC-02 30
Tabla 14. RUC-03 30
Tabla 15. RUC-04 31
Tabla 16. RUC-05 31
Tabla 17. RUC-06 31
Tabla 18. RUC-07 31
Tabla 19. RUC-08 32
Tabla 20. RUR-01 32
Tabla 21. RSF-01 33
Tabla 22. RSF-02 33
Tabla 23. RSF-03 33
Tabla 24. RSF-04 33
Tabla 25. RSF-05 34
Tabla 26. RSF-06 34
Tabla 27. RSF-07 34
Tabla 28. RSF-08 34
Tabla 29. RSF-09 35
Tabla 30. RSF-10 35
Tabla 31. RSF-11 35

Tabla 32. RSNF-01 36
Tabla 33. RSNF-02 36
Tabla 34. RSNF-03 36
Tabla 35. RSNF-04 36
Tabla 36. RH-01 37
Tabla 37. RH-02 37
Tabla 38. Requisitos mínimos rFactor 2 y especificaciones ordenador de prueba 48
Tabla 39. Especificaciones del dispositivo móvil de prueba 49
Tabla 40. PA-01 49
Tabla 41. PA-02 49
Tabla 42. PA-03 49
Tabla 43. PA-04 50
Tabla 44. PA-05 50
Tabla 45. PA-06 50
Tabla 46. Matriz de trazabilidad RSF-PA 51
Tabla 47. Costes de personal 53
Tabla 48. Costes de hardware 54
Tabla 49. Costes de software 54
Tabla 50. Coste total del proyecto 54
Tabla 51. Beneficios de la aplicación móvil en el primer año 55

1. INTRODUCCIÓN

En este apartado se explicarán los motivos por los que se decide llevar a cabo este Proyecto de Fin de Carrera, sus objetivos y la estructura que se ha seguido para redactar esta memoria.

1.1. MOTIVACIÓN

Durante los últimos años los juegos han estado en constante evolución. La industria del videojuego ha tenido que adaptarse tanto a jugadores casuales como a jugadores profesionales, con el auge de los eSports.

En particular, los juegos de conducción han tenido tal avance, son capaces de llegar a un nivel de realismo, que algunos ya no son considerados juegos, sino simuladores. No solamente por los gráficos que lucen, sino por la implementación de las físicas, el comportamiento de los coches, bastante parecido a lo que se puede experimentar en una pista de carreras real.

El simulador que se va a utilizar en este proyecto es rFactor 2. "El simulador de carreras abierto", desarrollado por Image Space Incorporated y ahora mantenido por Studio 397, se caracteriza por la facilidad para implementar modificaciones y añadidos al juego base.

A su vez, el uso de dispositivos móviles inteligentes es ahora mismo una rutina para todos. Con los sistemas operativos e interfaces que presentan es muy fácil que lleguen a prácticamente cualquier tipo de persona, ya esté familiarizado con las nuevas tecnologías o no.

Los dos sistemas operativos móviles predominantes son Android, desarrollado por Google y iOS, desarrollado por Apple. El primero de ellos se acerca a una cuota de mercado del 80%, por lo que es el elegido para el desarrollo de este proyecto.

El objetivo principal de este proyecto es la implementación de una aplicación Android que ayude al jugador a gestionar una sesión de simulación en rFactor 2 de forma fácil y

sencilla, con la mínima configuración posible, de modo que sirva por un lado como iniciación al mundo de los simuladores de carreras y por otro como un elemento más que ayude a la inmersión en la simulación en forma de Hud personalizado, sustituyendo en parte la información mostrada por el simulador en pantalla.

1.2. OBJETIVOS

A continuación se listan los objetivos que se deben alcanzar para la correcta realización del proyecto:

- Estudio del API público que proporciona rFactor 2 y creación de un plugin que obtenga la información necesaria del simulador.
- Diseño e implementación de un sistema de comunicación entre el plugin y la aplicación Android.
- Diseño e implementación de una interfaz adaptada a dispositivos móviles, lo más sencilla posible, que reciba la información y que la muestre de una manera acorde al simulador.
- Diseño de varias interfaces distintas para mostrar los datos del simulador, dando al usuario la posibilidad de elegir entre ellas.

1.3. ESTRUCTURA DE LA MEMORIA

En este apartado se listará el contenido de cada uno delos capítulos que componen la memoria del Proyecto de Fin de Carrera:

En el Capítulo 1, INTRODUCCIÓN, se hace una descripción del contexto del proyecto, con sus motivaciones y principales objetivos. También se hace una descripción de la estructura del documento.

En el Capítulo 2, ESTADO DEL ARTE, se analizan a fondo las tecnologías que serán necesarias para la realización del proyecto.

En el Capítulo 3, ANÁLISIS DEL SISTEMA, se hace un estudio de la funcionalidad necesaria en el sistema que se va a desarrollar, con el objetivo de definir los requisitos de usuario, de software y de hardware para después comenzar con el diseño y la implementación.

En el Capítulo 4, DISEÑO DEL SISTEMA, se define el diseño de los componentes del sistema, desde la arquitectura escogida hasta la especificación del sistema de comunicación que se va a usar entre plugin y aplicación móvil.

En el Capítulo 5, IMPLEMENTACIÓN Y PRUEBAS, se explica cómo se ha llevado a cabo la implementación de la funcionalidad requerida, a partir del análisis y el diseño, y se presentarán las soluciones que se han encontrado a los problemas que han ido surgiendo. También se describirán las pruebas que se han llevado a cabo para asegurar el correcto funcionamiento del sistema.

En el Capítulo 6, CONCLUSIÓN Y AMPLIACIONES FUTURAS, se recogerán las conclusiones posteriores a la realización del proyecto, así como una lista de posibles actualizaciones futuras. En este apartado también se incluye el desglose de los costes que forman el presupuesto.

En el Capítulo 7, BIBLIOGRAFÍA, se lista la bibliografía utilizada durante la realización del proyecto.

2. ESTADO DEL ARTE

En este apartado se va a presentar una visión general de las tecnologías que van a formar parte del proyecto, necesaria para una mejor comprensión antes de las fases de análisis y diseño.

2.1. ANDROID

Android es un sistema operativo para dispositivos móviles creado en 2003 por una pequeña empresa estadounidense llamada Android Inc, fundada por un antiguo y veterano ingeniero de Apple llamado Andy Rubin. Después de varios años de desarrollo, llamaron la atención de Google, que la adquirió en 2005 viendo que era lo que necesitaban para competir en el terreno de los móviles inteligentes con Microsoft y Blackberry [1].

En 2008, tras varios prototipos de hardware donde poder ejecutar el sistema operativo, se lanza al mercado la primera versión de Android junto con el terminal T-Mobile G1. Desde entonces han sido varias las versiones de Android que han ido viendo la luz, cada una de las principales con un nombre en clave que corresponde con un dulce y por orden alfabético.

Número de versión	Nombre clave	Lanzamiento
1.0	Android 1.0	23 septiembre 2008
1.1	Android 1.1	9 febrero 2009
1.5	Cupcake	27 abril 2009
1.6	Donut	15 septiembre 2009
2.0-2.1	Eclair	26 octubre 2009
2.2-2.2.3	Froyo	20 mayo 2010
2.3-2.3.7	Gingerbread	6 diciembre 2010
3.0-3.2.6	Honeycomb	22 febrero 2011
4.0-4.0.5	Ice Cream Sandwich	18 octubre 2011
4.1-4.3.1	Jelly Bean	9 julio 2012
4.4-4.4.4	KitKat	31 octubre 2013
5.0-5.1.1	Lollipop	12 noviembre 2014

6.0-6.1	Marshmallow	5 octubre 2015
7.0-7.1.2	Nougat	15 junio 2016
8.0	Oreo	21 agosto 2017

Tabla 1. Versiones de Android

El código fuente de Android está liberado bajo una licencia Apache [2], de código abierto y, gracias a esto, los fabricantes aprovechan para personalizarlo al máximo, por lo que la mayoría de dispositivos Android tienen una combinación de código abierto y software propietario, incluyendo el software requerido para acceder a los servicios de Google. Realmente son pocos los dispositivos que operan con la versión de Android pura, sin modificaciones.

Se trata de un sistema operativo con un kernel basado en el de Linux, encima del cual están los middleware, librerías y APIs, escritos en C, y después el framework Java [3] donde se ejecutan las aplicaciones.

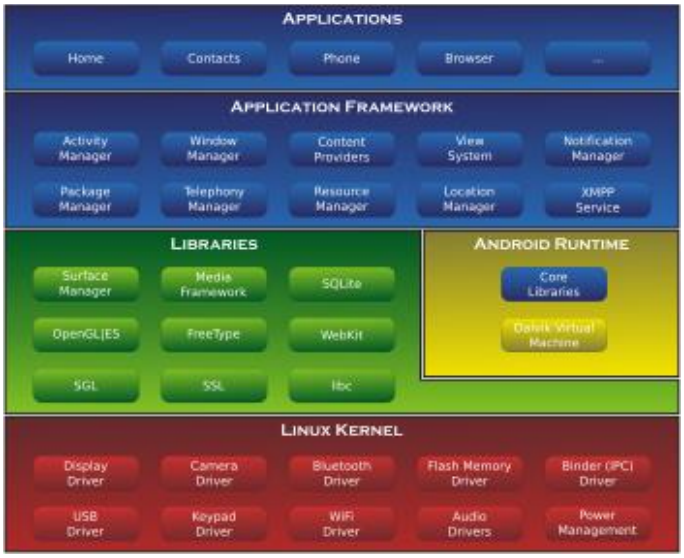


Figura 1. Arquitectura de Android

Hasta la versión 5.0, Android usaba Dalvik como máquina virtual, que ejecutaba código derivado del bytecode de Java. En 2015, Google anunció que en la siguiente versión de Android se cambiaría a una implementación de Java basada en OpenJDK [4].

En cuanto a las aplicaciones y su desarrollo, Google pone a disposición del desarrollador un SDK que incluye librerías, depurador, emulador, documentación,

muestras de código y tutoriales. Inicialmente, Google daba soporte como IDE a Eclipse usando el plugin ADT, sin embargo en 2014 la propia Google lanzaba Android Studio [5] como IDE oficial, basado en IntelliJ IDEA [6]. Una de sus principales características es el uso de Gradle [7], una herramienta para automatizar el proceso de construcción del proyecto cuyos puntos fuertes son una gran flexibilidad y un sólido sistema de gestión de dependencias.

Uno de los mayores problemas con los que se enfrentan los desarrolladores de aplicaciones es la compatibilidad, debido a la gran variedad de dispositivos y versiones que tiene Android. Para la compatibilidad entre las versiones, el propio SDK de Android ofrece muchas facilidades, haciendo que un mismo código pueda ser ejecutado sin fallos y con el mismo resultado en distintas versiones. Aun así, uno de las primeras decisiones a tomar antes de empezar a desarrollar es la versión mínima, para intentar abarcar el máximo de versiones y dispositivos posible. Sin embargo, en cuanto a los dispositivos, cada uno con funciones hardware distintas, es responsabilidad del desarrollador el controlar que dichas funciones estén disponibles. Por ejemplo, puede haber dispositivos que no tengan GPS, por lo que la aplicación tiene que estar preparada para funcionar sin acceder a esta funcionalidad del dispositivo.

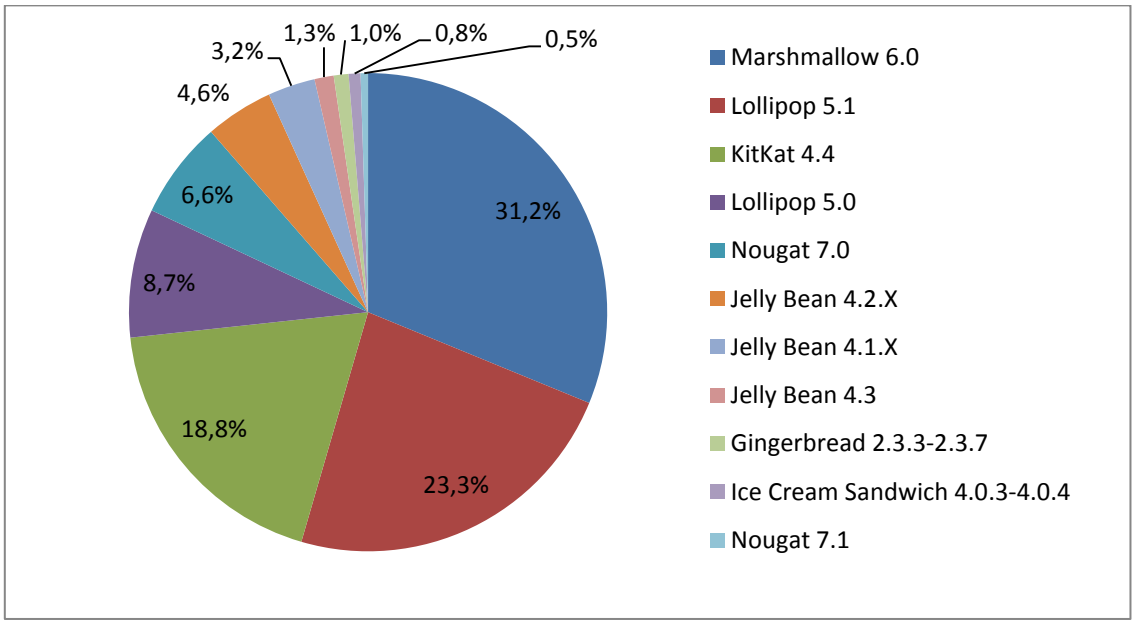


Figura 2. Distribución de versiones de Android en mayo de 2017

Otro punto a tener en cuenta de la compatibilidad son los tamaños de pantalla. La aplicación tiene que estar preparada para poder visualizarse correctamente en varios tamaños de pantalla y eso implica en muchas ocasiones implementar un diseño completamente distinto, ya que debido a la gran aceptación de Android, no solamente se encuentra disponible en teléfonos, sino también en tablets con un tamaño más grande de pantalla.

Para la distribución de aplicaciones existe una tienda online, Google Play Store [8], donde hay infinidad de aplicaciones de todos los tipos. Desde 2009, la cantidad de aplicaciones en Google Play Store se ha visto incrementada de 16.000 a 3.000.000 en junio de 2017.

Para publicar una aplicación es necesaria una licencia de desarrollador, que se adquiere al ingresar como desarrollador por 25$. Luego, en la consola de desarrollador hay disponibles herramientas estadísticas y de gestión de errores que facilitan la tarea del desarrollador.

2.1.1. Anatomía de una aplicación Android

Dentro de una aplicación de Android hay cuatro componentes principales: *Activities, Listeners, Services y Content Providers*. Todas las aplicaciones de *Android* están formadas por algunos de estos elementos o combinaciones de ellos.

Activity

Las *Activities* (o Actividades) son el elemento constituyente de Android más común. Para implementarlas se utiliza una clase por cada Actividad que extiende de la clase base *Activity* [9]. Cada clase mostrará una interfaz de usuario, compuesta por *Views* (o Vistas). Cada vez que se cambie de Vista, se cambiará de Actividad, como por ejemplo en una aplicación de mensajería que se tiene una Vista que muestra la lista de contactos y otra Vista para escribir los mensajes. Cuando cambiamos de Vista, la anterior queda pausada y puesta dentro de una pila de historial para poder retornar en caso necesario. También se pueden eliminar las Vistas del historial en caso de que no se necesiten más. Para pasar de vista en vista, *Android* utiliza una clase especial llamada *Intent*.

Un *Intent* [10] es un objeto mensaje y que, en general, describe qué quiere hacer una aplicación. Las dos partes más importantes de un *Intent* son la acción que se quiere realizar y la información necesaria que se proporciona para poder realizarla. Relacionado con los *Intents*, hay una clase llamada *IntentFilter* que es una descripción de qué *Intents* puede una gestionar un *Activity*. Mediante los *IntentFilters*, el sistema puede resolver *Intents*, buscando cuáles posee cada actividad y escogiendo aquel que mejor se ajuste a sus necesidades. El proceso de resolver *Intents* se realiza en tiempo real, lo cual ofrece dos beneficios:

- Las actividades pueden reutilizar funcionalidades de otros componentes simplemente haciendo peticiones mediante un *Intent.*
- Las actividades pueden ser remplazadas por nuevas actividades con *IntentFilters* equivalentes.

Listeners

Los Listeneres se utilizan para reaccionar a eventos externos (por ejemplo, una llamada). Los Listeners no tienen UI (User Interface), pero pueden utilizar el servicio NotificationManager para avisar al usuario. Para lanzar un aviso no hace falta que la aplicación se esté ejecutando, en caso necesario, Android la iniciará si se activa el Listeners por algún evento.

Content Providers

En Android, las aplicaciones pueden guardar su información en ficheros, BBDD *SQLite*, etc., pero en caso de que lo que se quiera sea compartir dicha información con otras aplicaciones, lo necesario es un *Content Provider*. Un *Content Provider* es una clase que implementa un conjunto estándar de métodos que permite a otras aplicaciones guardar y obtener la información que maneja dicho *Content Provider.*

Android Manifest

En Android existe un archivo *XML* llamado *AndroidManifest* que, aunque no forme parte del código principal de la aplicación, es necesario para su correcto funcionamiento. Este archivo es el fichero de control que le dice al sistema qué tiene que hacer con todos los

componentes anteriormente mencionados en este apartado que pertenecen a una aplicación en concreto.

2.1.2. Ciclo de vida de una aplicación Android

Cada aplicación de Android corre en su propio proceso, el cual es creado por la aplicación cuando se ejecuta, y permanece hasta que la aplicación deja de trabajar o el sistema necesita memoria para otras aplicaciones. Una característica fundamental de Android es que el ciclo de vida de una aplicación no está controlado por la misma aplicación sino que lo determina el sistema a partir de una combinación de estados como pueden ser qué aplicaciones están funcionando, qué prioridad tienen para el usuario y cuánta memoria queda disponible en el sistema. De esta manera, Android sitúa cada proceso en una jerarquía de "importancia" basada en los estados comentados, como se puede ver a continuación.

- Un proceso en primer plano es uno que se requiere para lo que el usuario está actualmente haciendo. Se considera en primer plano si:
    - Está ejecutándose una Actividad perteneciente a la pantalla con la que el usuario está interactuando.
    - Está ejecutando un *BroadcastReceiver*.
    - Está ejecutándose un servicio.
- Un proceso visible es aquel que contiene una Actividad que es visible al usuario mediante la pantalla, pero no en primer plano (está pausada). Este proceso solo se eliminará en caso de que sea necesario para mantener ejecutándose los procesos en primer plano.
- Un proceso de servicio es aquel que contiene un servicio que ha sido inicializado. No son directamente visibles al usuario y el sistema los mantendrá a no ser que no pueda servir los dos anteriores.
- Un proceso en *background* es aquel que acoge una actividad que no está actualmente visible al usuario. Mientras que dichos procesos implementen bien su propio ciclo de vida, el sistema puede eliminarlos para dar memoria a cualquiera de los 3 procesos anteriores.

- Un proceso vacío es aquel que no contiene ningún componente activo de ninguna aplicación. La única razón para mantener dicho proceso es para mejorar sus inicializaciones posteriores a modo de caché.

Para comprender mejor el ciclo de vida de una aplicación de Android, en la siguiente figura se muestra el diagrama de flujo de dicho ciclo, mencionando también los métodos que se llaman durante el transcurso del mismo.

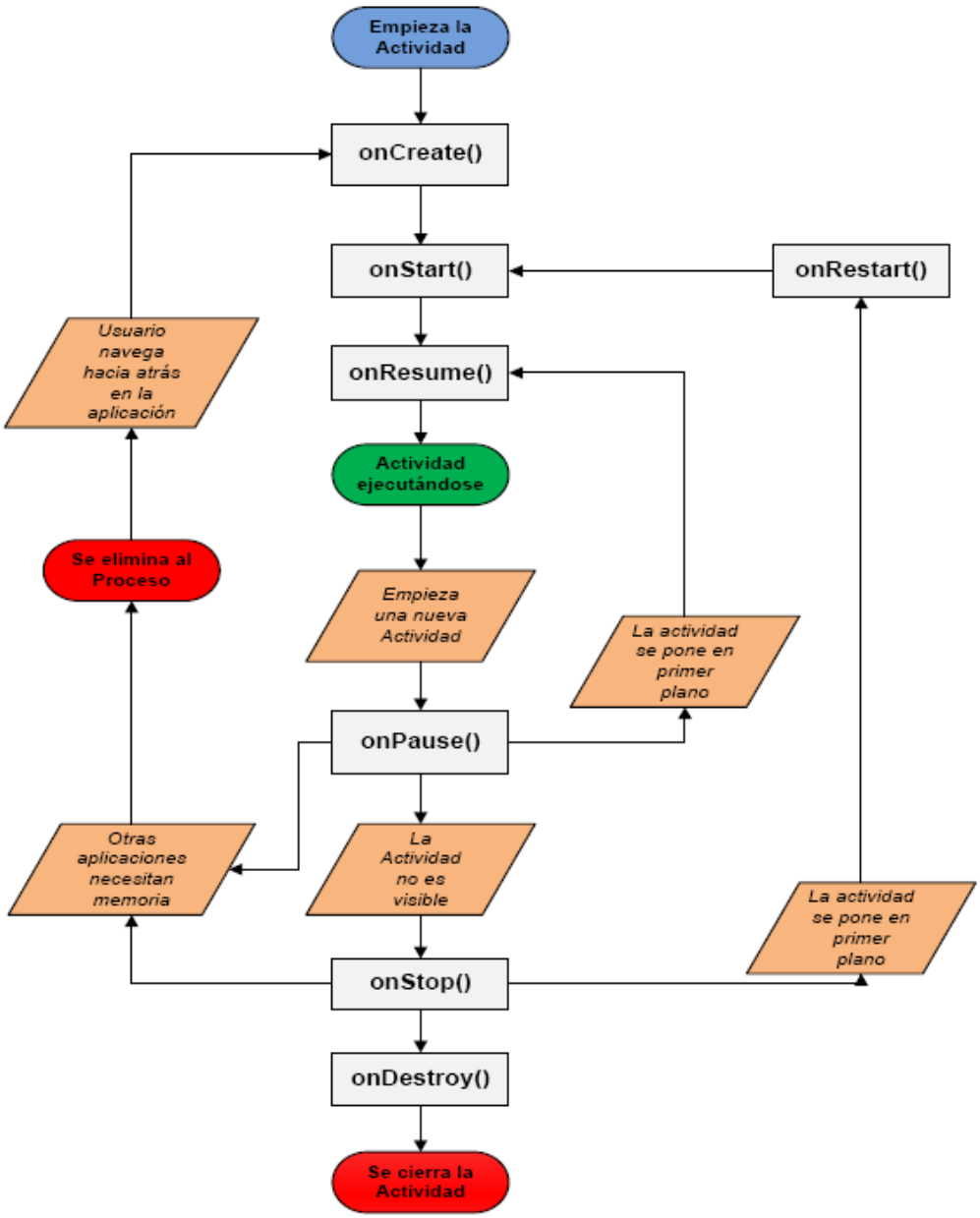


Figura 3. Ciclo de vida de una Actividad Android

2.2. RFACTOR 2

rFactor 2 [11] es un simulador de conducción desarrollado por Image Space Incorporated (ISI). Sucesor del exitoso rFactor, fue publicado en 2013 para el sistema operativo Windows. Como su predecesor, y como la mayoría de juegos de carreras desarrollados por ISI, está diseñado para ser modificado, facilitando la implementación de mods que mejoren la experiencia de juego. Dicha facilidad residía en rFactor en una estructura sencilla de carpetas y ficheros, la mayor parte de los cuales podía abrirse y modificarse con cualquier editor de texto. En rFactor 2 ese sistema de carpetas se mantiene, pero también se añade un sistema de empaquetamiento de mods que hace que su creación e instalación y distribución sea mucho más cómoda.

Tanto rFactor como rFactor 2 han recibido muchos elogios por su altamente avanzado y preciso sistema de simulación de dinámica de vehículos. La mayor parte de su código proviene derivado de rFactor Pro, el cual es el preferido de equipos profesionales de carreras, ingenieros del automóvil y centros de simulación, siendo utilizado por la mayoría de constructores de Fórmula 1 y NASCAR [12].

El juego base cuenta con algunos circuitos y coches reales, de los que posee licencia de uso, y de otros totalmente inventados. Sin embargo, la mayor parte de contenido que se puede disfrutar en el juego proviene de los mods. Se pueden encontrar infinidad de coches, circuitos, herramientas, plugins, etc. Por ejemplo, una herramienta muy popular entre los simracers es el plugin Motec [13], que permite el análisis de la telemetría del coche a posteriori.

El sistema de juego es muy sencillo. Mediante menús en pantalla podemos ver todos los circuitos y coches que tenemos instalados y seleccionar los que queremos, para después configurar la sesión de simulación. Podemos configurarla solo con entrenamientos libres o añadirle también sesión de clasificación y de carrera. Todas las sesiones se pueden limitar por tiempo, se puede definir la hora del día y se pueden escoger normas como el consumo de combustible y de neumáticos (normal o aumentado por 2, por 3, etc.) y hasta la meteorología.

La reciente adquisición de rFactor 2 por parte de la desarrolladora Studio 397 y la publicación en la plataforma de juegos Steam [14], con su herramienta de creación y

distribución de contenido Steam Workshop, auguran un buen futuro para el desarrollo de mods para rFactor 2.

Una vez vista un poco la historia y una descripción general de rFactor, a continuación se describirá su sistema de plugins.

2.2.1. Sistema de plugins

rFactor 2 ofrece la posibilidad de implementar plugins que pueden cambiar el comportamiento del simulador o añadirle funcionalidad. Estos plugins consisten en una dll que hay que colocar en la carpeta rFactor2/BinXX/Plugins, siendo XX igual a 32 o 64, dependiendo de si la dll se ha desarrollado para arquitectura de 32 bits o 64 bits. Cuando se ejecuta el juego, se encarga automáticamente de ejecutar las dll que haya en dicha carpeta.

La estructura que tiene que tener la dll la podemos saber gracias la documentación y a un ejemplo que hay disponible en la propia página web de rFactor 2 [15]. Después de descargarlo, podemos ver las clases y ficheros siguientes:

Fichero	InternalsPlugin.hpp
Descripción	Fichero de cabeceras que define todas las estructuras de datos.
Interfaces, clases y estructuras contenidas	
Nombre	**Descripción**
`struct TelemInfo`	Contiene datos de la telemetría en tiempo real del coche, tales como la vuelta en la que se encuentra, la marcha, las revoluciones del motor, las entradas del piloto (giro del volante, acelerador y freno) y muchas más.
`struct ScoringInfo`	Contiene datos acerca del circuito y la sesión en la que nos encontramos (entrenamientos, clasificación o carrera). También contiene datos de todos los coches que se encuentran en el circuito, pero solamente a nivel de clasificaciones y tiempos.
`struct GraphicsInfo`	Contiene información de los gráficos que se están mostrando en pantalla, por si se quisieran añadir algunos propios.
`class InternalsPlugin`	Define la interfaz con la que se comunica el simulador en ejecución. No contiene implementación, solamente la interfaz.

Tabla 2. Fichero InternalsPlugin.hpp

Clase	InternalsPlugin
Método	**Descripción**
`void Startup(long)`	Llamado al iniciarse el simulador. Recibe como parámetro la versión del simulador.
`void Shutdown()`	Llamado al cerrarse el simulador.
`void StartSession()`	Llamado cuando se carga un circuito y cada vez que se avanza de sesión.
`void EndSession()`	Llamado cada vez que la sesión acaba y cuando se abandona el circuito.
`void EnterRealtime()`	Llamado cuando entramos en el coche y salimos a pista.
`void ExitRealtime()`	Llamado cuando salimos del coche.
`long WantsTelemetryUpdates()`	Llamado al iniciarse el simulador, sirve para registrar el plugin para recibir actualizaciones de telemetría. Ofrece la posibilidad de registrarlo para obtener sólo la telemetría de nuestro coche o la de todos los coches, aunque esta última no es tan precisa.
`void UpdateTelemetry (TelemInfo)`	Una vez estemos con nuestro coche en pista, en este método se reciben las actualizaciones de telemetría. Según la documentación, esta actualización se produce cada vez que se genera un nuevo frame gráfico, pero no más de 90 veces por segundo. Por esto, también se sugiere que el código que se implemente debe ser eficiente y que lo único que se debe hacer es leer datos y enviarlos a otra aplicación o dispositivo.
`bool WantsScoringUpdates ()`	Llamado al iniciarse el simulador, con él podemos registrar el plugin para recibir información del circuito y la sesión.
`void UpdateScoring (ScoringInfo)`	Cuando está cargado el circuito, estemos o no con nuestro coche en pista, se llama a este método para actualizar la información de clasificaciones y tiempos. Ya que no es una información que cambie con tanta frecuencia como la telemetría, este método es llamado solamente 5 veces por segundo.
`bool WantsGraphicsUpdates()`	Llamado al iniciarse el simulador para registrarse para recibir información de los gráficos de la pantalla.
`void UpdateGraphics (GraphicsInfo)`	Llamado una vez que se carga un circuito y cada vez que se genera un frame gráfico.

Tabla 3. Clase InternalsPlugin

Fichero	Example.hpp
Descripción	Fichero de cabeceras de la clase principal.
Interfaces, clases y estructuras contenidas	
Nombre	**Descripción**
`class ExampleInternalsPlugin : InternalsPlugin`	Define la clase principal del plugin, ya que hereda de la clase. `InternalsPlugin.`

Tabla 4. Fichero Example.hpp

Fichero	Example.cpp
Descripción	Fichero donde se implementa la clase principal, la clase `ExampleInternalsPlugin`. En el plugin que se vaya a desarrollar debe haber una clase como ésta, ya que será la puerta de comunicación con el simulador.

Tabla 5. Fichero Example.cpp

3. ANÁLISIS DEL SISTEMA

El objetivo de este apartado es definir qué debe hace la aplicación que se va a desarrollar.

Después de haber analizado el funcionamiento de rFactor 2 con su sistema de plugins, el sistema que se va a desarrollar contará con dos partes bien diferenciadas:

- Plugin: se añadirá al simulador y se registrará para recibir las actualizaciones oportunas. Cuando reciba dichas actualizaciones, seleccionará los datos más relevantes y los enviará a la aplicación móvil.
- Aplicación móvil: se conectará al plugin y recibirá los datos del simulador que le envíe para mostrarlos en la pantalla del dispositivo, haciendo para ello alguna transformación o algún cálculo si fuese necesario.

El producto del análisis que se va a realizar en este apartado son los requisitos de usuario, que es en lo que se basará el posterior diseño y la implementación del proyecto.

3.1. DIAGRAMA DE ESTADOS DEL SIMULADOR

Después de haber analizado las clases e interfaces que componen el API de rFactor 2 y de haber visto en funcionamiento el plugin de ejemplo, se puede definir el siguiente diagrama de estados por los que pasa el simulador:

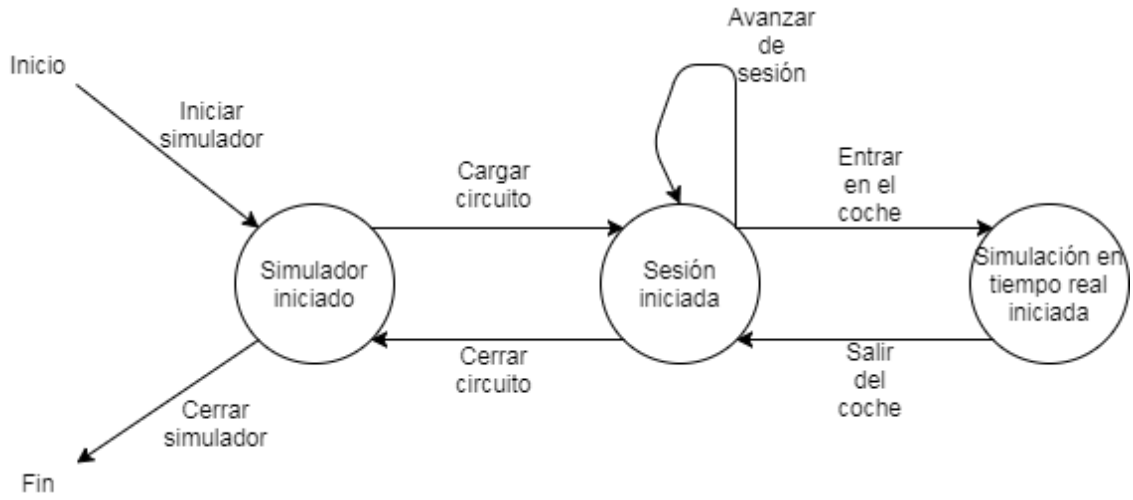


Figura 4. Diagrama de estados del simulador

- Simulador iniciado: en este estado el simulador ha ejecutado todos los plugins que tiene añadidos y los ha registrado para recibir las actualizaciones que tienen definidas.
- Sesión iniciada: se reciben actualizaciones de circuito y sesión.
- Simulación en tiempo real iniciada: Se reciben actualizaciones de telemetría del coche. También se reciben actualizaciones de circuito y sesión.

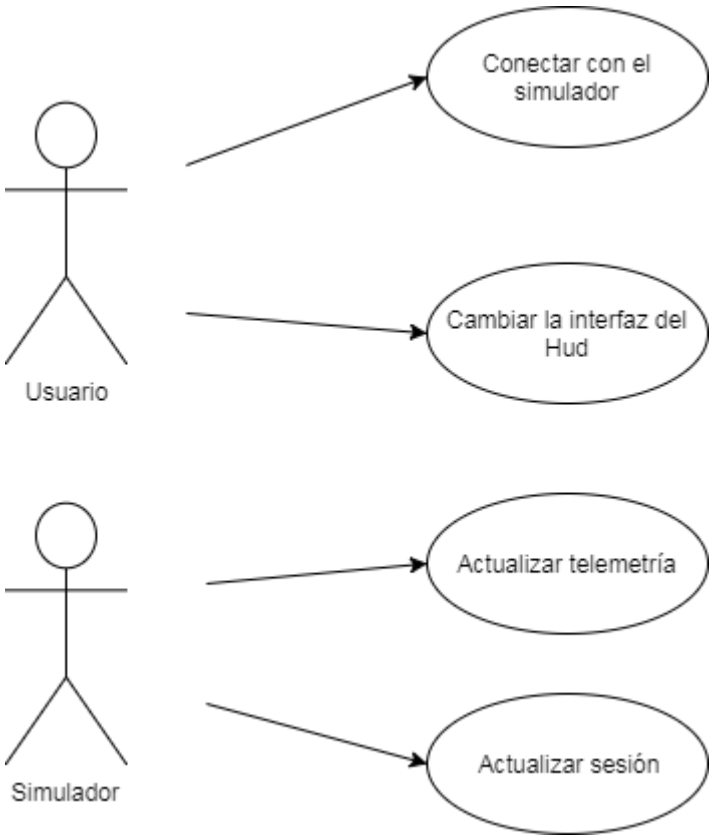
3.2 CASOS DE USO

A continuación se van a definir los casos de uso del sistema que se va a desarrollar. El objetivo es identificar los requisitos funcionales estructurados en torno a los usuarios y otros actores que pueden interactuar con el sistema. Se va a usar la siguiente plantilla:

CU-XX	Título
Objetivo	
Actores	
Precondiciones	
Postcondiciones	
Escenario principal	
Escenarios alternativos	

Tabla 6. Plantilla de Casos de Uso

3.2.1 Descripción gráfica de los casos de uso



3.2.2. Descripción textual de los casos de uso

CU-01	Conectar con el simulador
Objetivo	Conectar la aplicación móvil con el simulador.
Actores	Usuario
Precondiciones	El plugin debe haber sido añadido al simulador, el simulador debe estar en ejecución y la aplicación móvil instalada en un dispositivo. Tanto el dispositivo móvil como el ordenador deben estar conectados a la misma red.
Postcondiciones	La aplicación móvil recibe actualizaciones desde el simulador.
Escenario principal	1. El usuario pulsa el icono de la aplicación en su dispositivo móvil y aparece la pantalla principal. 2. El usuario introduce la dirección ip del ordenador donde se está ejecutando el simulador en el cuadro de texto identificado como “Dirección Ip”. 3. El usuario pulsa el botón “Conectar”. 4. Aparece una pantalla con mensaje notificando el éxito de la conexión.

Escenarios alternativos	4a. La aplicación informa al usuario de que ha habido un error en la conexión. 1. El usuario corrige algún posible error en la dirección ip introducida. 2. El usuario pulsa el botón "Conectar". 3. Aparece una pantalla con mensaje notificando el éxito de la conexión.

Tabla 7. CU-01, Conectar con el simulador

CU-02	Cambiar la interfaz del Hud
Objetivo	El usuario decide cambiar la apariencia de la pantalla del Hud.
Actores	Usuario
Precondiciones	La aplicación debe estar instalada en el dispositivo.
Postcondiciones	La siguiente vez que se acceda a una sesión de simulación se muestra la interfaz elegida.
Escenario principal	1. El usuario pulsa el icono de la aplicación en su dispositivo móvil y aparece la pantalla principal. 2. El usuario pulsa el botón identificado como "Configuración". 3. La aplicación muestra una lista con las interfaces disponibles. 4. El usuario selecciona una de las interfaces y la aplicación vuelve a la pantalla principal.
Escenarios alternativos	5. El usuario conecta la aplicación al simulador según el CU-01. 6. El usuario inicia una sesión de simulación. La aplicación muestra el Hud seleccionado.

Tabla 8. CU-02, Cambiar la interfaz del Hud

CU-03	Actualizar telemetría
Objetivo	El simulador envía una actualización de telemetría.
Actores	Simulador
Precondiciones	El plugin debe haber sido añadido al simulador, el simulador debe estar en ejecución, en una sesión activa y con el coche en pista. La aplicación móvil instalada en un dispositivo y conectada al simulador.
Postcondiciones	La información mostrada en el Hud cambia.
Escenario principal	1. El simulador envía una actualización de telemetría al plugin. 2. El plugin selecciona los datos relevantes de la telemetría y los envía a la aplicación móvil. 3. La aplicación móvil cambia la información mostrada.
Escenarios alternativos	

Tabla 9. CU-03, Actualizar telemetría

CU-04	Actualizar sesión
Objetivo	El simulador envía una actualización de información de la sesión.
Actores	Simulador

Precondiciones	El plugin debe haber sido añadido al simulador, el simulador debe estar en ejecución, en una sesión activa y con el coche en pista. La aplicación móvil instalada en un dispositivo y conectada al simulador.
Postcondiciones	En la aplicación parece una nueva pantalla del Hud.
Escenario principal	1. El simulador envía una actualización de sesión al plugin. 2. El plugin selecciona los datos relevantes de la sesión y los envía a la aplicación móvil. 3. La aplicación móvil muestra una nueva pantalla del Hud, sin información.
Escenarios alternativos	

Tabla 10. CU-04, Actualizar sesión

3.3. REQUISITOS DE USUARIO

En este apartado se van a describir los requisitos de usuario, obtenidos de conversaciones y comentarios del cliente. Primero se listan los requisitos de capacidad que son los que sirven para especificar la funcionalidad que el sistema es fundamental que tenga. Después estarán los requisitos de restricción para definir cómo se debe construir el software.

Para la definición de requisitos se usará la siguiente plantilla:

Identificador: RUC/RUR-XX			
Prioridad		**Fuente**	
Necesidad		**Estabilidad**	
Descripción			

Tabla 11. Plantilla de los requisitos de Usuario

Cada requisito de usuario viene definido por los siguientes atributos:

- Identificador: compuesto por un prefijo y un número, debe ser único para identificar unívocamente al requisito. El prefijo es RUC para requisitos de capacidad y RUR para requisitos de restricción.
- Necesidad: puede ser esencial, deseable y opcional. Indican la importancia que le da el cliente y si puede estar abierto a negociación.

- Prioridad: dirigido al desarrollador, puede ayudar a decidir la planificación del desarrollo. Puede tomar los valores alta, media y baja.
- Estabilidad: indica la probabilidad con la que el requisito puede cambiar, ya sea por su descripción o necesidad. Puede tomar los valores estable e inestable. Es útil para saber en qué partes del desarrollo prestar más atención para posibles cambios.
- Fuente: el origen del requisito. Suele ser el cliente o alguna documentación.
- Descripción: explicación del requisito, de manera clara y precisa.

3.3.1. Requisitos de capacidad

Identificador: RUC-01			
Prioridad	Alta	**Fuente**	Cliente
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	Poder recibir actualizaciones de telemetría y de sesión en tiempo real desde el simulador.		

Tabla 12. RUC-01

Identificador: RUC-02			
Prioridad	Alta	**Fuente**	Cliente
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	Contar con una conexión entre el plugin de rFactor 2 y la aplicación móvil.		

Tabla 13. RUC-02

Identificador: RUC-03			
Prioridad	Alta	**Fuente**	Cliente
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	Enviar actualizaciones del simulador desde el plugin de rFactor 2 a la aplicación móvil.		

Tabla 14. RUC-03

Identificador: RUC-04			
Prioridad	Media	**Fuente**	Cliente
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	Tener distintos tipos de Hud para mostrar la información del simulador. Se piden tres tipos de Hud: • Estilo Fórmula 1 • Estilo Fórmula 1 con leds dedicados al DRS • Estilo GT		

Tabla 15. RUC-04

Identificador: RUC-05			
Prioridad	Media	**Fuente**	Cliente
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	Contar con un menú para seleccionar los distintos tipos de Hud.		

Tabla 16. RUC-05

Identificador: RUC-06			
Prioridad	Alta	**Fuente**	Cliente
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	Diseñar la aplicación móvil de forma que en un futuro se puedan añadir más tipos de Hud de forma sencilla.		

Tabla 17. RUC-06

Identificador: RUC-07			
Prioridad	Alta	**Fuente**	Cliente
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	Informar del estado de la conexión entre el plugin y la aplicación móvil.		

Tabla 18. RUC-07

Identificador: RUC-08			
Prioridad	Alta	**Fuente**	Cliente
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	Mostrar las actualizaciones del simulador en la aplicación móvil.		

Tabla 19. RUC-08

3.3.2. Requisitos de restricción

Identificador: RUR-01			
Prioridad	Alta	**Fuente**	Cliente
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	La conexión entre el plugin y la aplicación móvil se iniciará introduciendo la dirección ip del ordenador donde se esté ejecutando el simulador.		

Tabla 20. RUR-01

3.4. REQUISITOS DE SOFTWARE

En este apartado se definen los requisitos de software. Dichos requisitos surgen del análisis de los requisitos de usuario identificados en el apartado anterior. Para definirlos se usará la misma tabla, únicamente cambiando los identificadores que ahora pasan a ser RSF para requisitos funcionales y RSNF para requisitos no funcionales.

3.4.1. Requisitos funcionales

Identificador: RSF-01			
Prioridad	Alta	**Fuente**	RUC-02
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	Añadir un botón en la pantalla principal para iniciar la conexión.		

Tabla 21. RSF-01

Identificador: RSF-02			
Prioridad	Alta	**Fuente**	RUC-04
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	Crear interfaz para el Hud estilo Fórmula 1.		

Tabla 22. RSF-02

Identificador: RSF-03			
Prioridad	Alta	**Fuente**	RUC-04
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	Crear interfaz para el Hud estilo Fórmula 1 con leds para mostrar el estado del DRS.		

Tabla 23. RSF-03

Identificador: RSF-04			
Prioridad	Alta	**Fuente**	RUC-04
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	Crear interfaz para el Hud estilo GT.		

Tabla 24. RSF-04

Identificador: RSF-05			
Prioridad	Media	**Fuente**	RUC-05

Necesidad	Esencial	**Estabilidad**	Estable
Descripción	Añadir un botón a la pantalla principal para acceder al menú de selección de Huds.		

Tabla 25. RSF-05

Identificador: RSF-06			
Prioridad	Media	**Fuente**	RUC-05
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	Crear un menú que muestre los distintos tipos de Hud disponibles.		

Tabla 26. RSF-06

Identificador: RSF-07			
Prioridad	Media	**Fuente**	RUC-05
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	En el menú de Huds, distinguir cuál es el que está seleccionado actualmente.		

Tabla 27. RSF-07

Identificador: RSF-08			
Prioridad	Alta	**Fuente**	RUC-05
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	Guardar la selección de Hud de forma persistente.		

Tabla 28. RSF-08

Identificador: RSF-09			
Prioridad	Alta	**Fuente**	RUC-07
Necesidad	Esencial	**Estabilidad**	Estable

Descripción	La aplicación móvil mostrará un error si se produce algún error en la conexión.

Tabla 29. RSF-09

Identificador: RSF-10			
Prioridad	Alta	**Fuente**	RUC-08
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	La aplicación móvil mantendrá actualizado el Hud en pantalla según se vayan recibiendo datos desde el plugin.		

Tabla 30. RSF-10

Identificador: RSF-11			
Prioridad	Alta	**Fuente**	RUR-01
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	Añadir un campo de texto en la pantalla principal para introducir la dirección IP a la que conectarse.		

Tabla 31. RSF-11

3.4.2. Requisitos no funcionales

Identificador: RSNF-01			
Prioridad	Alta	**Fuente**	RUC-01
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	El plugin debe registrarse para recibir actualizaciones de telemetría y sesión, pero no de gráficos.		

Tabla 32. RSNF-01

Identificador: RSNF-02			
Prioridad	Alta	**Fuente**	RUC-02
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	La conexión entre el plugin y la aplicación móvil se realizará mediante sockets.		

Tabla 33. RSNF-02

Identificador: RSNF-03			
Prioridad	Alta	**Fuente**	RUC-03
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	El formato de envío de los datos desde el plugin hasta la aplicación móvil debe ser ligero para no perjudicar el rendimiento.		

Tabla 34. RSNF-03

Identificador: RSNF-04			
Prioridad	Alta	**Fuente**	RUC-03
Necesidad	Esencial	**Estabilidad**	Estable
Descripción	La aplicación móvil debe hacer los cálculos y las transformaciones mínimas al mostrar la información para no perjudicar el rendimiento.		

Tabla 35. RSNF-04

3.5. REQUISITOS DE HARDWARE

En este apartado se recogen los requisitos de hardware necesarios para el funcionamiento del sistema.

Identificador: RH-01	
Descripción	La aplicación móvil se va a desarrollar para el sistema operativo Android 4.1, por lo que es necesario que el dispositivo pueda ejecutar esa versión o una posterior.

Tabla 36. RH-01

Identificador: RH-02	
Descripción	Es necesario que el dispositivo móvil disponga de conexión mediante Wifi.

Tabla 37. RH-02

4. DISEÑO DEL SISTEMA

En este apartado se definirá el diseño arquitectónico del sistema, el cual servirá más adelante para el proceso de implementación.

4.1. ARQUITECTURA DEL SISTEMA

Para el diseño de este sistema podríamos estar hablando de un sistema de tiempo real, con la única diferencia de que no se necesitaría responder a la generación de los eventos. Por lo tanto podríamos definir el sistema completo como un sistema de monitorización. Sin embargo, el sistema puede describirse con una arquitectura Modelo-Vista-Controlador que, como veremos a continuación, aporta algunas ventajas.

4.1.1. Arquitectura Modelo-Vista-Controlador

La arquitectura Modelo-Vista-Controlador (MVC) es una arquitectura de software que surgió junto con las primeras interfaces gráficas. Se compone de tres elementos bien diferenciados:

- Modelo: es la representación de la información con la cual el sistema opera y gestiona todos los accesos a dicha información.
- Vista: presenta el Modelo en un formato adecuado para presentarla al usuario y proporciona la interacción con el mismo.
- Controlador: es el intermediario entre la Vista y el Modelo. Responde a eventos generados por el usuario en la Vista y realiza peticiones al Modelo cuando se hace alguna solicitud sobre la información. Cuando se recibe nueva información, se encarga de comunicársela a la Vista para su presentación.

Se trata de una arquitectura que se basa en las ideas de reutilización de código y separación de conceptos, las cuales son unas características con las que se busca facilitar el desarrollo y el mantenimiento de las aplicaciones.

Para el sistema que se pretende desarrollar, los tres elementos están distribuidos entre el simulador rFactor 2, el plugin que se implementará para su API, las interfaces gráficas de la aplicación Android y el sistema de comunicación entre plugin y aplicación.

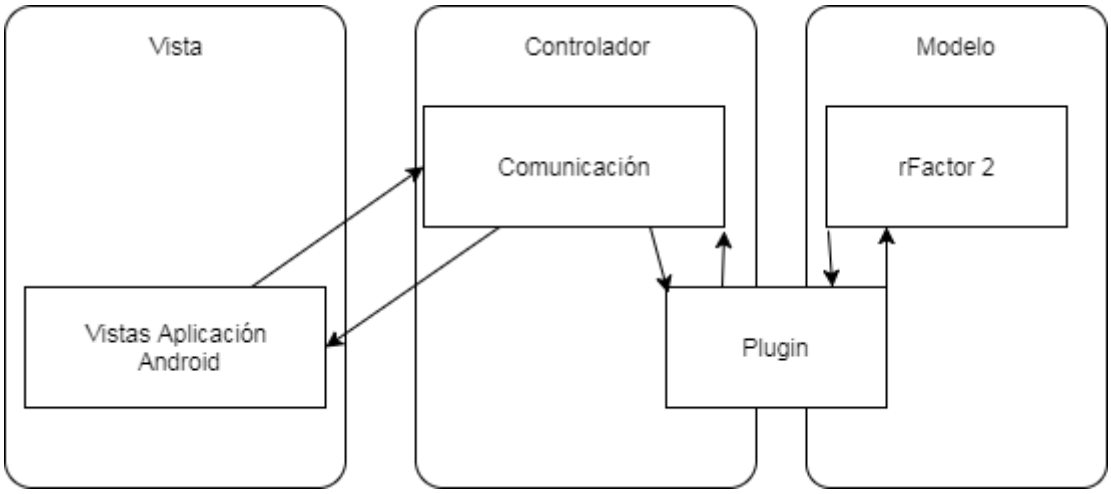


Figura 5. Arquitectura MVC del sistema

Además, se podría decir que la aplicación Android es un sistema MVC por sí sola, ya que se pueden distinguir estos tres elementos bien diferenciados que se describen en el siguiente apartado.

4.1.2. MVC en una aplicación Android

En una aplicación Android los tres elementos de una arquitectura Modelo-Vista-Controlador pueden implementarse aprovechando algunos componentes propios del API de Android:

- El modelo puede implementarse mediante acceso a servicios web con la clase *AsyncTask* [16], acceso a base de datos mediante la clase *SQLiteDatabase* [17], etc. mientras que los distintos elementos del modelo pueden ser de la clase *Object* [18] de Java.
- La vista es un elemento muy diferenciado en una aplicación Android, ya que se implementa con ficheros XML en los que se define una estructura (*layout*) de objetos gráficos como botones, imágenes, etiquetas, etc. [19]
- El controlador normalmente será un objeto de la clase *Activity* o Fragment [20], los cuales tienen una estrecha relación con los *layouts*, ya que a cada Actividad o

Fragmento que esté en activo en la aplicación, al crearlo se le asigna un *layout* para su visualización.

4.2. DISEÑO DETALLADO

En este apartado se detallará el diseño del sistema a un nivel más bajo. A continuación se describirán las clases principales que forman cada componente del sistema:

- Plugin
    - AndroidHud: clase principal del plugin que se añadirá al simulador, debe heredar de *InternalsPlugin*. Recibe las actualizaciones del mismo, tanto de telemetría como de sesión.
    - Comms: utilizada por la clase *AndroidHud*, encapsula un socket para la comunicación con la aplicación Android.
- Aplicación Android
    - MainActivity: es la actividad que representa la primera pantalla que se carga cuando se inicia la app. Permite la conexión con el plugin.
    - ConnectedWaitingActivity: en esta actividad se representa el estado en el que se ha producido la conexión y se está esperando a que comience una sesión en el simulador.
    - RealTimeActivity: es la actividad con la que se representa la pantalla en la que se reciben las actualizaciones en tiempo real desde el simulador.
    - HudListActivity: es la actividad con la que se mostrará la lista de Huds disponibles en la aplicación para que el usuario pueda elegir el que desea usar.
    - Comms: es la clase análoga a la del plugin, proporciona la comunicación con el plugin encapsulando un socket.
    - RealTime: en esta clase se encapsula un objeto de la clase *Comms* y es donde se reciben todos los mensajes. Después, se encarga de su conversión a un objeto que se pueda usar en el resto de la aplicación y lo distribuye al observador que tenga registrado.
    - ScoringInfo: clase que forma parte del modelo. Contiene información sobre la vuelta actual en la que se encuentra el coche.

- SessionInfo: forma parte del modelo y contiene información sobre el circuito. Típicamente, se recibirá solamente una vez cuando empiece cada sesión.
- TelemetryInfo: también es parte del modelo. Contiene la información en tiempo real de la telemetría del coche.
- HudController: es una interfaz usada por la actividad *RealTimeActivity*. La idea es que, aunque se tengan distintos tipos de Hud, todos tengan la misma interfaz, ya que todos deben tratar la misma información, solamente deben mostrarla de forma distinta.
- F1HudController: clase que implementa la interfaz *HudController*, específica para el Hud de Fórmula 1.
- F1DrsHudController: implementa la interfaz *HudController*, será parecida a F1HudController pero mostrará de forma distinta los datos acerca del DRS.
- GTHudController: implementa la interfaz HudController, le dará otro aspecto al Hud, distinto del de Fórmula 1.

Realmente, en la aplicación el usuario solamente puede navegar entre las actividades *MainActivity* y *HudListActivity*. La navegación entre las demás se realiza de forma automática según se reciban eventos desde el simulador.

Otros componentes secundarios en la aplicación Android son los controladores de cada elemento de la telemetría en tiempo real. Cada uno estará definido por una interfaz para poder implementarse de distinta forma en distintas clases usando un patrón Strategy [21]. Todas estas interfaces serán usadas por las clases que implementan la interfaz *HudController*, siendo los componentes que controlan cada parte del Hud. Los controladores diseñados son los siguientes:

- SpeedController: controla la vista en la que se muestra la velocidad actual del coche.
- FuelController: gestiona la vista en la que se representa la cantidad de gasolina. En algunas implementaciones podría hacer alguna predicción de consumo.

- GearController: gestiona la marcha actual que tiene el coche.
- RpmController: controla las revoluciones del motor.
- SectorsController: gestiona la vista en la que se muestra el estado de las banderas en los diferentes sectores del circuito.
- DrsController: controla la vista que representa el estado del DRS.

4.2.1. Sistema de comunicación

Uno de los componentes más importantes del sistema a desarrollar es el sistema de comunicación entre el plugin del simulador y la aplicación Android, ya que sin él no se cumpliría el objetivo principal que es ver la información del simulador en un dispositivo móvil. En este apartado se definirán los mensajes que el plugin enviará a la aplicación Android.

Cada mensaje estará compuesto de dos partes: el nombre del evento y una lista de pares clave-valor, aunque esta lista es opcional y solamente la tienen algunos mensajes. Los mensajes son los siguientes:

- StartSession: enviado al inicio de cada sesión. Enviará una lista de mensajes con estas claves y los valores de este tipo:
    - CircuitLen-Double: la longitud del circuito.
    - Session-Integer: la sesión que acaba de empezar. 1 a 3=entrenamientos, 5=clasificación, 6=calentamiento, 7=carrera.
    - MaxLaps-Integer: opcional, indica el máximo de vueltas que se pueden dar en la sesión. Se recibirá solamente en las sesiones que están acotadas por número de vueltas, no por tiempo.
- RTScoring: envía las actualizaciones en tiempo real de la posición en el circuito. En este caso solamente envía un par clave-valor.
    - LapDist-Double: longitud del circuito que lleva recorrida el coche.
- RTTelem: envía la telemetría en tiempo real del coche. Los valores que envía en la lista son:
    - Lap-Integer: la vuelta actual en la que se encuentra el coche.

    - Fuel-Double: la cantidad de gasolina que tiene el coche.
    - Rpm-Double: las revoluciones por minuto del motor.
    - MaxRpm: el máximo de revoluciones por minuto que puede llegar a conseguir el motor.
    - Gear-Integer: la marcha actual que lleva el coche.
    - Speed-Double: la velocidad actual del coche.
    - Sectors: esta clave llevará asociada tres valores 0-1, cada uno para indicar un sector del circuito. El valor 1 representa que hay una bandera amarilla en el sector.
    - Drs-Integer: el estado del DRS. 0=inactivo y sin posibilidad de activarlo, 1=inactivo pero se podrá activar en la siguiente zona de DRS, 2=inactivo pero activación posible, 3=activo.
- RTExit: enviado cuando se abandona el coche pero se mantiene la sesión activa. No envía lista, solamente indica que no se recibirán actualizaciones de telemetría por el momento.
- EndSession: enviado cuando se termina una sesión. Indica que no se recibirán más actualizaciones de telemetría hasta que se vuelva a recibir el mensaje StartSession.
- End: se envía cuando se cierra el simulador.

4.3. OTRAS DECISIONES DE DISEÑO

Al desarrollar usando Java es común utilizar paralelización con hilos. En Android, hay que tener mucho cuidado con el hilo en el que se va a ejecutar determinado código, ya que tiene dos restricciones principales: los elementos gráficos solamente pueden manipularse desde el hilo que se crearon y no se puede ejecutar código de redes y conexiones en el hilo de la interfaz gráfica. Por esto, se decide que cada vez que se vaya a ejecutar código de conexiones se haga mediante un hilo usando la clase Java *Thread* (java.lang.Thread) y que, cuando se reciban datos y se vayan a enviar a los componentes gráficos se haga usando el método *runOnUIThread* de la clase *Activity*).

Para el desarrollo de una vista compuesta por una lista, en Android se suele utilizar la clase *RecyclerView* (android.support.v7.widget.RecyclerView), ya que dispone de una buena gestión de la memoria reciclando, como indica su nombre, las vistas más

pequeñas que componen los ítems de la lista. Este reciclaje consiste en reutilizar las vistas que no se están mostrando porque se haya hecho *scroll* para mostrar otras partes de la lista y así consumir menos memoria. Este componente se usará en la lista de Huds.

Para el diseño de las interfaces gráficas se usará la propia herramienta gráfica de Android Studio. Además, se usará como *layout* base para todas las interfaces un componente llamado *ConstraintLayout* (android.support.constraint.ConstraintLayout), disponible desde la versión 2.3.3 de Android Studio siendo parte de una librería de soporte que hace que se pueda usar en versiones anteriores de Android, hasta la 2.3.3 Gingerbread. Este *layout* base hace que la colocación de los componentes gráficos sea flexible y se adapte fácilmente a los distintos tamaños de pantalla.

5. IMPLEMENTACIÓN Y PRUEBAS

5.1. IMPLEMENTACIÓN

En la implementación del plugin se ha tenido en cuenta los métodos del API de rFactor 2 por los que se reciben actualizaciones continuamente y los que solamente envían eventos puntuales.

En el método *UpdateTelemetry* solamente se recibirán actualizaciones si el coche está en pista, así que en este método solo se implementa la lógica de envío a la aplicación del mensaje *RTTelem*. Sin embargo, en el método *UpdateScoring* se están enviando actualizaciones desde que se carga la sesión y para este sistema que se está desarrollando solamente son necesarios cuando también se está con el coche en pista., así que es necesario implementar una lógica para ver si hay que enviar estas actualizaciones o no. Además, desde este método se van a enviar los mensajes *StartSession* y *RTScoring*, ya que ambos envían la información a partir de la misma estructura de datos del API de rFactor 2, la clase *ScoringInfo*. Por esto también es necesario otra lógica para saber qué mensaje enviar.

El mensaje *StartSession* solamente se va a enviar una vez al principio de la sesión así que, se usa una variable booleana *sentStartingSessionInfo* que se pondrá a *true* en el método que también se llama *StartSession*, del API de rFactor. Una vez enviado, se puede volver a poner a *false* para no enviarse más hasta el siguiente inicio de sesión.

Para el mensaje *RTScoring* se hace de forma parecida, con la variable booleana *sendRealTimeInfo*, que se pone a *true* en el método *EnterRealtime* y a *false* en el método *ExitRealtime*. Mientras esta variable esté a *true* el mensaje se enviará a la aplicación móvil.

En la implementación de la aplicación móvil, se decide simplificar el desarrollo de las actividades y que sean ellas mismas las que se encarguen de recibir los eventos de la pantalla, implementando la interfaz *OnClickListener* en lugar de implementar componentes separados solamente para esta funcionalidad. Esto se decide porque ninguna de las actividades va a tener una lógica muy complicada, que de tenerla

seguramente sí se habría optado por implementar componentes separados para los *listeners* de los eventos.

Para el sistema de comunicación también se ha optado por unas interfaces escuchadoras de eventos, *RealTimeListener* y *RealTimeEventListener*. Las actividades implementarán la interfaz *RealTimeEventListener*, se registrarán como escuchadores en su método *onCreate* y dirigirán el flujo de la aplicación hacia otra interfaz cuando reciban un evento. Mientras, las clases que implementen la interfaz *HudController*, también implementarán la interfaz *RealTimeListener* para actualizar la interfaz gráfica con las actualizaciones en tiempo real.

El sistema de almacenamiento para la configuración de la aplicación es las preferencias compartidas, *SharedPreferences*, del propio API de Android. La cantidad de datos que se van a guardar no es muy grande, por lo que no es necesario una base de datos.

Como se ha comentado en el apartado de diseño, se pretende que con una sola actividad para la interfaz de tiempo real, *RealTimeActivity*, se puedan visualizar distintas interfaces, es decir, los distintos tipos de Hud. Esto se consigue con el patrón Factory [21] y con la interfaz *HudController*, convertida en clase abstracta, que actúa como factoría de los distintos Huds. Usando el método *setContentView* de la clase *Activity*, cada *Controller* concreto puede configurar la actividad con la interfaz gráfica que le corresponda, y con el método *findViewById* puede acceder a sus componentes gráficos para configurar los *Controllers* específicos de cada parte de la interfaz. Se puede ver este funcionamiento en el siguiente diagrama de secuencia.

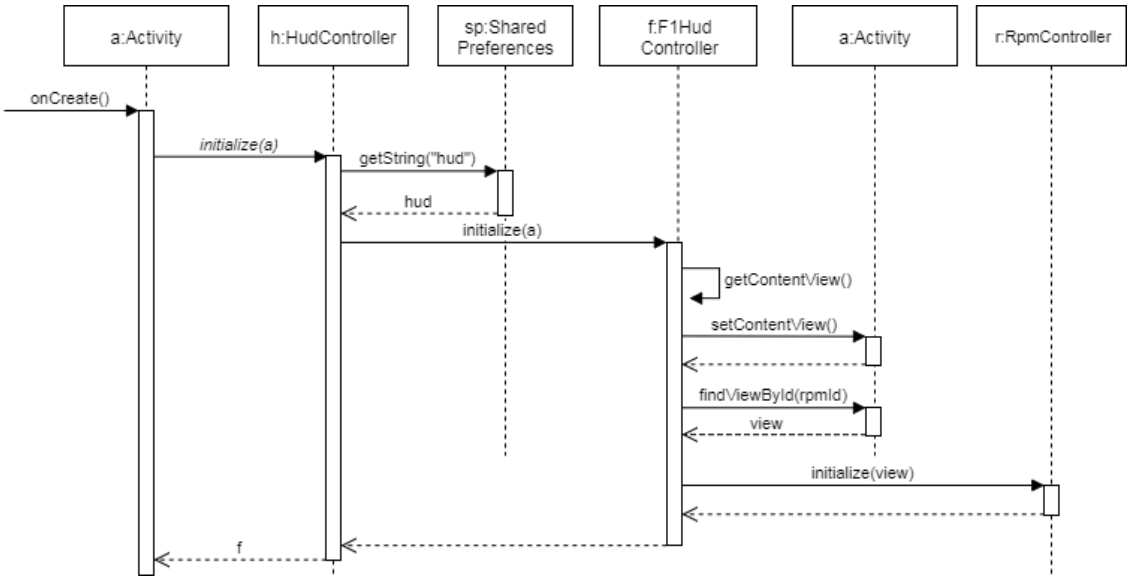


Figura 6. Diagrama de secuencia de la creación de una interfaz gráfica

En las aplicaciones Android, para poder tener acceso a determinadas funcionalidades, es necesario pedir unos permisos al sistema. En el caso de esta aplicación solamente es necesario el permiso *INTERNET*, para poder establecer conexiones, así que se añade este permiso al fichero *manifest* de la aplicación.

En el conjunto del sistema y durante el proceso de desarrollo e implementación, se decide implementar un sistema de logs que faciliten la implementación y la depuración del programa. Para el caso del plugin se opta por mostrar los mensajes que se envían en una consola que se está ejecutando a la vez que el simulador, además de unos ficheros que recogen los mensajes enviados y que después de la ejecución se pueden consultar. En la aplicación Android, se emplea la clase estática *Log* (android.util.Log) y sus métodos *e()* y *d()*, que sirven para trazar errores e información de depuración respectivamente, y que junto con la consola Android que está disponible en Android Studio, se convierten en una herramienta imprescindible para un desarrollador de aplicaciones Android. Aun así, al tratarse de un sistema que durante buena parte de su ejecución emplea informaciones y actualizaciones en tiempo real, la tarea de depuración no siempre resulta fácil, ya que los mensajes se generan a gran velocidad y puede ser complicado seguirlos. Algunas veces han resultado más útil los ficheros de log generados y consultados a posteriori que las trazas en tiempo real.

5.2. PRUEBAS

En este apartado se describirán las especificaciones del hardware que se ha usado para probar el sistema, así como las pruebas de aceptación del sistema que garantizan que funcione correctamente.

5.2.1. Hardware de prueba

A continuación se listan las especificaciones del ordenador en el que se ha probado el plugin desarrollado, comparándolas con los requisitos mínimos que pide el simulador rFactor 2. Es importante al menos cumplir con los requisitos mínimos para un correcto funcionamiento del simulador y que no repercuta en el funcionamiento del sistema.

	Requisitos mínimos	Ordenador de prueba
Sistema Operativo	Windows 7, 8, 8.1 o 10	Windows 7
Procesador	2.8 GHz Intel Core 2 Duo o 3.0 GHz AMD Athlon II x2	Intel Core i7 2600 a 3.4 GHz
Memoria RAM	4 GB	8GB
Tarjeta Gráfica	nVidia GTS 450 o AMD Radeon 5750	nVidia GTX 560 Ti
DirectX	9.0c	11

Tabla 38. Requisitos mínimos rFactor 2 y especificaciones ordenador de prueba

En la siguiente tabla se pueden ver las especificaciones del dispositivo móvil que se ha empleado durante las pruebas del sistema. Además de este dispositivo móvil, se han usado distintos tamaños de pantalla desde 4 hasta 10 pulgadas, gracias a los simuladores que ofrece Android Studio, para asegurar la correcta visualización de las interfaces.

Dispositivo móvil de prueba	
Fabricante	Samsung
Modelo	S3 Mini
Versión de Android	4.1.2 Jelly Bean

Versión del Kernel	3.0.31-1514264
Procesador	1GHz Dual core
Pantalla	4 pulgadas
Memoria RAM	1GB
Almacenamiento interno	8GB
Otras características	3G, Wifi, NFC

Tabla 39. Especificaciones del dispositivo móvil de prueba

5.2.2. Pruebas de aceptación

PA-01	
Descripción	Conectar la aplicación móvil con el plugin.
Resultado	Éxito.

Tabla 40. PA-01

PA-02	
Descripción	Comenzar una simulación con el Hud estilo Fórmula 1.
Resultado	Éxito.

Tabla 41. PA-02

PA-03	
Descripción	Comenzar una simulación con el Hud estilo Fórmula 1 con leds especiales para DRS.
Resultado	Éxito.

Tabla 42. PA-03

PA-04	
Descripción	Comenzar una simulación con el Hud estilo GT.
Resultado	Éxito.

Tabla 43. PA-04

PA-05	
Descripción	Conectar la aplicación móvil al plugin y desactivar la conexión Wifi del dispositivo.
Resultado	Éxito.

Tabla 44. PA-05

PA-06	
Descripción	Comenzar una simulación en tiempo real, con la aplicación móvil conectada y sacar el coche a pista.
Resultado	Éxito.

Tabla 45. PA-06

5.3.2. Matriz de trazabilidad RSF-PA

RSF \ PA	01	02	03	04	05	06
01	X					
02		X				
03			X			
04				X		
05		X	X	X		
06		X	X	X		
07		X	X	X		
08		X	X	X		
09					X	
10						X
11	X					

Tabla 46. Matriz de trazabilidad RSF-PA

6. CONCLUSIÓN Y AMPLIACIONES FUTURAS

En este apartado se presentarán las conclusiones a las que se ha llegado después de la realización de este proyecto y se comentarán algunas ampliaciones que se podrán llevar a cabo en un futuro.

6.1 CONCLUSIÓN

Está claro que hoy en día los móviles son un compañero inseparable y gran parte de esto es por la gran cantidad de aplicaciones que hay disponibles en el mercado. Buena parte de culpa de esa gran cantidad de aplicaciones la tienen los entornos de desarrollo tan sencillos e intuitivos, las APIs, que publican las grandes empresas como Google y Apple, y la gran cantidad de información que hay al respecto. Por esto, recomiendo a cualquiera que tenga un mínimo interés en la programación que pruebe el desarrollo de aplicaciones móviles.

La verdadera dificultad del desarrollo de aplicaciones es la originalidad. El trabajo de crear una nueva aplicación se ha convertido en una tarea más sencilla para el programador, teniéndose que centrar el desarrollo en la funcionalidad que se ofrece para destacar entre todas las aplicaciones, lo cual no es fácil. Es la idea que pone en marcha la aplicación la que hace que al final sea atractiva para un buen número de personas o que sea verdaderamente útil para un perfil de usuario muy específico.

En este proyecto, el reto para mí estaba en ahondar un poco más en la programación Android, de la que ya conocía una pequeña parte pero no en gran medida, y sobre todo en la creación del plugin para rFactor 2. Como comentario personal, he querido hacer un desarrollo parecido al de este proyecto desde que supe que se podía hacer. Y poder hacerlo como proyecto de fin de carrera, hacerlo sobre algo que me gusta y lleva gustándome mucho tiempo, ha sido realmente enriquecedor y productivo.

6.2. OBJETIVOS CUMPLIDOS

La totalidad de los objetivos listados en el primer apartado se ha cumplido de forma satisfactoria. Tanto el plugin de rFactor 2 como la aplicación móvil cumplen con la funcionalidad propuesta y después de haberlos probado con un pequeño número de simracers, sería posible su distribución tanto en Google Play Store como en alguna web dedicada a rFactor 2.

6.3. PRESUPUESTO

En este apartado se lleva a cabo el cálculo de los costes derivados del proyecto. Se van a separar los costes en costes de personal, hardware y software en distintos subapartados.

Para el cálculo de los costes se han tenido en cuentas las siguientes consideraciones:

- La fecha de inicio del proyecto es el 1 de julio de 2017 y la fecha de fin el 22 de septiembre de 2017. El total de días, sin festivos ni fines de semana, es de 64. Si a esto restamos 9 días de período vacacional en agosto, nos queda un total de 55 días laborables.
- La jornada laboral comprende 8 horas diarias.
- El total de horas es de 440 horas.

6.3.1. Costes de personal

Etapa del proyecto	Categoría	Coste por hora (€/h)	Número horas	Coste Total (€)
Análisis	Analista	40	80	3.200
Diseño	Diseñador	40	80	3.200
Diseño gráfico	Diseñador gráfico	25	20	500
Codificación	Programador	30	150	4.500
Pruebas	Programador	20	50	1.000
Documentación	Analista	20	60	1.200
TOTAL			**440**	**13.600**

Tabla 47. Costes de personal

6.3.2. Costes de hardware

Concepto	Coste(€)	Unidades	Coste Total (€)
Samsung S3 Mini	200	1	200
PC Dell XPS	1.200	1	1.200
TOTAL			**1.400**

Tabla 48. Costes de hardware

6.3.3. Costes de software

Concepto	Coste(€)	Unidades	Coste Total (€)
Licencia desarrollador Android	22	1	22
Licencia Visual Studio	45	1x3 meses	135
rFactor 2	30	1	30
TOTAL			**187**

Tabla 49. Costes de software

6.3.4. Coste total

Concepto	Coste Total (€)
Recursos humanos	13.600
Hardware	1.400
Software	187
Total sin IVA	**15.187**
Beneficio (20%)	3.037
IVA (21%)	3.827
TOTAL	**22.052**

Tabla 50. Coste total del proyecto

6.3.5. Beneficios

Los beneficios que va a generar la aplicación móvil provienen únicamente de las descargas que se realicen en Google Play Store. Se ha decidido que el precio de la aplicación sea de 1€ y se han estimado unas 2.000 descargas mensuales, lo que significa 24.000 descargas anuales.

Concepto	Precio(€)	Unidades	Beneficio Total (€)
Descargas en Google Play Store	1	24000	24000
TOTAL			**24.000**

Tabla 51. Beneficios de la aplicación móvil en el primer año

6.4. AMPLIACIONES FUTURAS

A continuación se exponen algunas ideas que pueden ser incluidas en la aplicación en futuras actualizaciones, algunas de ellas incluso están definidas y preparadas desde los requisitos de usuario:

- Añadir más tipos de Hud. El desarrollo de la aplicación se ha hecho de forma que no sea difícil añadir más diseños de Hud. Unos ejemplos podrían ser unos diales clásicos o imitar los de algún coche real en particular.
- Añadir un Hud personalizable, dando al usuario la posibilidad de elegir qué elementos mostrar en la interfaz y colocarlos donde quiera.
- Desarrollo para iOS, el otro gran sistema operativo para móviles. La mayor parte de aplicaciones se suelen desarrollar para ambas plataformas, ya que son las que más cuota de mercado tienen.

Además, la realización de este proyecto puede dar pie al desarrollo de otros parecidos, una vez estudiada la API de rFactor 2, como por ejemplo:

- Sistema de análisis de telemetría, para ayudar al jugador a configurar el coche.
- Realizar un sistema Hud con hardware, independiente de aplicaciones móviles.

7. BIBLIOGRAFÍA

[1] Historia de Android (Julio 2017)

https://www.androidcentral.com/android-history

[2] Apache (Julio 2017)

http://www.apache.org/licenses/

[3] Java (Julio 2017)

http://www.oracle.com/technetwork/java/index.html

[4] OpenJDK (Julio 2017)

http://openjdk.java.net/

[5] Android Studio (Agosto 2017)

https://developer.android.com/studio/index.html

[6] IntelliJ IDEA (Agosto 2017)

https://www.jetbrains.com/idea/

[7] Gradle (Agosto 2017)

https://gradle.org/

[8] Google Play Store (Septiembre 2017)

https://play.google.com/store

[9] Activity (Agosto 2017)

https://developer.android.com/reference/android/app/Activity.html

[10] Intent (Agosto 2017)

https://developer.android.com/reference/android/content/Intent.html

[11] rFactor 2 (Julio 2017)

https://www.studio-397.com/

[12] Offermars, Marcel. “rFactor: Full Steam Ahead!”. Planet Marrs (Julio 2017)

http://www.planetmarrs.net/rfactor-full-steam-ahead/

[13] Motec (Agosto 2017)

http://www.motec.com/

[14] rFactor 2 en Steam (Septiembre 2017)

http://store.steampowered.com/app/365960/rFactor_2/

[15] Recursos de desarrollo para rFactor 2 (Julio2017)

https://www.studio-397.com/modding-resources/

[16] AsyncTask (Agosto 2017)

https://developer.android.com/reference/android/os/AsyncTask.html

[17] SQLiteDatabase (Agosto 2017)

https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html

[18] Object (Agosto 2017)

https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html

[19] Layouts en Android (Agosto 2017)

https://developer.android.com/guide/topics/ui/declaring-layout.html

[20] Fragment (Agosto 2017)

https://developer.android.com/reference/android/app/Fragment.html

[21] Freeman, M.; Bates, B.; Sierra, K; Robson, E. “First Design Patterns: A Brain-Friendly Guide”